Econometrica, Vol. 69, No. 5 (September, 2001), 1261–1281

STOCHASTIC ALGORITHMS, SYMMETRIC MARKOV PERFECT EQUILIBRIUM, AND THE 'CURSE' OF DIMENSIONALITY

BY ARIEL PAKES AND PAUL MCGUIRE[1]

This paper introduces a stochastic algorithm for computing symmetric Markov perfect equilibria. The algorithm computes equilibrium policy and value functions, and generates a transition kernel for the (stochastic) evolution of the state of the system. It has two features that together imply that it need not be subject to the curse of dimensionality. First, the integral that determines continuation values is never calculated; rather it is approximated by a simple average of returns from past outcomes of the algorithm, an approximation whose computational burden is not tied to the dimension of the state space. Second, iterations of the algorithm update value and policy functions at a single (rather than at all possible) points in the state space. Random draws from a distribution set by the updated policies determine the location of the next iteration's updates. This selection only repeatedly hits the recurrent class of points, a subset whose cardinality is not directly tied to that of the state space. Numerical results for industrial organization problems show that our algorithm can increase speed and decrease memory requirements by several orders of magnitude.

KEYWORDS: Dynamic games, stochastic algorithms, curse of dimensionality.

APPLIED ANALYSIS OF DYNAMIC INDUSTRIAL organization (I.O.) problems is limited by the burden of computing equilibrium policies. These are typically obtained by computing a fixed point to a mapping of a function whose domain is a subset of $\mathbf{R}^n$, where n is the number of state variables in the problem. In I.O. problems n typically equals the number of state variables per agent times the (maximum) number of agents (ever) active. If each state variable takes on K distinct values and we impose symmetry on the policy functions (so that each agent's policy is the same function of its own and its competitors' states), then the number of possible state vectors, or the dimension of the fixed point calculation, grows geometrically in the number of agents and exponentially in the number of states per agent. This is one source of the "curse of dimensionality" in computing equilibria. The other is in the integral determining the continuation value for each agent active at each point; the number of terms in this sum grows in the same manner.

We introduce a stochastic algorithm that may circumvent these computational problems for Markov Perfect models (Maskin and Tirole (1988, 1994)) in which the state vector evolves as an ergodic Markov Process. This is because our algorithm (i) never attempts to obtain accurate policies on the entire state space, and (ii) never calculates continuation values; rather it uses an average of the value of outcomes from past iterations to approximate them.

[1]We thank the referees, G. Chamberlain, D. Fudenberg, G. Gowrisankaran, K. Judd, D. Pollard, J. Rust, and especially a co-editor for helpful comments, and the NSF (Grant SBR95-12106) for financial support.

Recall that in a finite state ergodic Markov Process every sample path will, in finite time, wander into the recurrent class of points, say R, and once in R will stay within it forever (see, for e.g., Freedman (1983, Ch. 1)). Thus to analyze the impacts of an event from an initial condition in R all we require is knowledge of the policies *on* R.

Our algorithm is *asynchronous*, it only calculates policies for a single point at each iteration (its location). The policies of each active agent at that point are obtained from a simple *single agent* maximization problem and maximize that iteration's estimate of the agent's continuation value. These policies set the distribution of the next iteration's state. A random draw from that distribution is taken and used to update the algorithm's location. This process eventually confines itself to selecting points in R. Depending on the economics of the problem, the number of points in R need not grow in the number of state variables (in n) at all.

The algorithm uses an average of past outcome to estimate the continuation values needed to determine policies. The estimates associated with the current iteration's location are updated by treating the random draw used to update the location of the algorithm as a Monte Carlo draw from the integral determining continuation values. That draw is evaluated and then averaged with the values of draws obtained from the same location in prior iterations to determine the new estimate of continuation values. Though this is both faster and has less memory requirements than explicitly integrating over possible future outcomes, it is also less precise (particularly in early iterations). This generates a tradeoff between the computational burden per point, and the number of iterations needed for a given level of precision. Since the precision of the estimate does not (necessarily) depend on the dimension of the integral being estimated, while the cost of doing the summation explicitly does, the larger is n the more we expect the tradeoff to favor our procedure.

Our rules for updating continuation values are similar to the rules used in some of the recent economic literature on learning (both in the context of single agent intertemporal decision problems and in the context of matrix games; examples include Fudenberg and Levine (1999), Lettau and Uhlig (1999), and Sargent (1993). Moreover, just as in that literature, our algorithm can be endowed with a behavioral interpretation by assuming that agents actually used our rules to form estimates of continuation values and then chose their policies to maximize the estimated values. We differ from that literature in dealing with games whose state vectors evolve over time and in focusing on how the learning algorithm can be used to help compute Nash equilibria (instead of on its behavioral or limit properties).

We do not know of general conditions which insure, a priori, that our algorithm will converge to the desired policies. We can, however, use conditions that insure convergence *has* occurred to build a stopping rule for the algorithm (which is the most we expect from applying fixed point algorithms to mappings that are not contractions; see Judd (1999) and the literature cited there). Also the computational advantages of our algorithm vary with the structure of the problem (e.g. with n and with the cardinality of the recurrent class). However, numerical results indicate that our algorithm is free of convergence problems and can convert problems that would have taken years on the current generation of supercomputers to problems that can be done in a few hours on our work station.

The next section outlines a Markov Perfect model of industry dynamics. This both helps to focus the subsequent discussion and provides a framework for computing examples. Section 2 considers the burden of computing equilibrium policies using pointwise backward solution algorithms. Section 3 introduces our algorithm and Section 4 contains numerical results that provide an indication of its power. A closing section considers related issues and an Appendix illustrates how easy it is to program our algorithm.

1. A SIMPLE MODEL

This section outlines a model of industry dynamics due to Ericson and Pakes (1995; henceforth EP) that is easy to work with.[2] In the model firms invest to explore profit opportunities. Profits in any period depend on the firm's and its competitors' level of quality or efficiency.

We let i index levels of efficiency, and assume $i \in \mathscr{Z}^+$, the positive integers. Let $s_i \in \mathscr{Z}^+$ be the number of firms with efficiency level i, so the vector $s = [s_i; i \in \mathscr{Z}^+]$ is the "market structure." Profits for a firm at i when the market structure is s are given by $\pi(i, s)$. Extensions in which i is a vector are discussed below.

$\pi(\cdot)$ can vary with demand and cost primitives, and with an equilibrium assumption.[3] Given $\pi(\cdot)$ an incumbent has two choices. It chooses whether to exit or remain active and if it remains active it chooses an amount of investment. If it exits it receives a sell-off value of ϕ dollars (and never reappears). If it invests x it incurs a cost of cx and has a probability distribution of improvements in i that is stochastically increasing in x.

Thus if we let β be the discount rate, and $p(i', s' \mid x, i, s)$ provide the firm's perceptions of the joint probability that its own efficiency in the next period will be i' and the industry structure will be s' conditional on (x, i, s), the Bellman

[2] For more on the model and extensions to it (largely made by others), see Pakes (2000).

[3] The publicly available program for computing our dynamic equilibria has three examples of $\pi(\cdot)$ (each programmed up to parameters set by the user). To access a description of, and code for, this algorithm (as well as a number of auxiliary programs designed to help analyze its results), either download the needed directories from Ariel Pakes' Harvard web page, or FTP to "econ.yale.edu", use "anonymous" as login, and your own user-name as "password." Then change directory to "pub/mrkv-eqm" and copy all needed files. The "read.me" file, or Pakes (2000), will start you off.

equation for the value of the firm $[V(i,s)]$ is

$$V(i,s) = \max\left\{\phi, \pi(i,s) + \sup_{(x\geq 0)}\left[-cx + \beta\sum V(i',s')p(i',s'\,|\,x,i,s)\right]\right\}. \tag{1}$$

The max operator determines if the continuation value (the expression after the comma) is greater than the sell-off value (ϕ). If not the firm shuts down. If so the firm chooses investment ($x \geq 0$). x determines the probabilities of increments in the firm's state over the period.

We assume that this increment can be written as a difference of two independent random variables, i.e.

$$i_{t+1} - i_t \equiv \nu_t - \zeta_t.$$

ν represents the outcome of the firm's investment process and has probabilities given by a family $\{p(\cdot\,|\,x),\ x \in \mathscr{R}^+\}$ that is stochastically increasing in x. ζ is an exogenous random variable with probabilities $\mu(\zeta)$. Its precise interpretation depends on the structure of the profit function but it typically represents common demand or supply shocks (e.g. competition from outside the industry, or factor prices). The ν's of different firms are independent, but the realization of ζ is common across firms. Both ν and ζ are non-negative integer valued random variables with finite support; $\nu = 0$ if $x = 0$ (a firm cannot advance without some investment), and $\mu(0) > 0$ as is $p(0\,|\,x)$ for all finite x.

Thus if $\hat{s}_i$ is the vector providing the states of the competitors of a firm at state i when the industry structure is s, and $q[\hat{s}'_i\,|\,i,s,\zeta]$ provides the firm's perceived probability that the states of its competitors in the next period will be $\hat{s}'_i$ conditional on a particular value of ζ,

$$\begin{aligned} p(i' = i^*,\ s' = s^*\,|\,x,i,s) \\ = \sum_{\zeta} p(\nu = i^* - i - \zeta\,|\,x)q[\hat{s}'_i = s^* - e(i^*)\,|\,i,s,\zeta]\mu(\zeta), \end{aligned} \tag{2}$$

where $e(i)$ is a vector that puts one in the ith slot and zero elsewhere (so that $s^* - e(i^*)$ gives the states of next year's competitors if $s' = s^*$ and $i' = i^*$). Note that $q[\cdot\,|\,i,s,\zeta]$ embodies the incumbent's beliefs about entry and exit.

For simplicity we assume there is only one potential entrant a period who pays an amount x_e ($> \beta\phi$) to enter, and enters one period later at state $\omega^e \in \Omega^e \subset \Omega$ with probability $p^e(\cdot)$. The entrant only enters if the value of entering is greater than x_e.[4]

That describes the primitives of the model. At an equilibrium the perceived distributions of future industry structures (the $q[\hat{s}'_i\,|\,i,s,\zeta]$) used to determine

[4]Different entry models are easy to accommodate provided the distribution of i's at which entry occurs is fixed over time. That is, the "ability" of entrants must progress at the same pace as the "ability" of the outside alternative; else entry would eventually go to zero and stay there.

the policies of a firm at any (i, s) equals the objective distribution of industry structures generated by the investment, entry, and exit policies of the competitors of the firm at (i, s). EP prove that provided their A.1 to A.7 hold:

(i) A MPE consisting of value, policy, and transition probabilities exists.

(ii) Any equilibrium has:

- a *finite* upper bound, say $\bar{n}$, to the number of firms that are ever simultaneously active;
- finite upper and lower bounds to observe values of "i".

(iii) Any equilibrium generates a (time homogeneous) finite state Markov chain for market structures [for $\{s_t\}$] with a transition kernel, say $Q[\cdot \mid \cdot]$, that is *ergodic*.

(ii) insures that only a finite number of (i, s) vectors will ever be observed. If 1 and K are the lower and upper bounds for "i", $\Omega \equiv \{1, \ldots, K\}$, $S \equiv \{s = [s_1, \ldots, s_k] : \Sigma s_j \leq \bar{n} < \infty\}$, and $s_0 \in S$, then $\forall t \ (i_t, s_t) \in \Omega \times S$ with $\#\Omega \times S < \infty$. I.e., the discrete state space, our conditions on the transition probabilities, and the upper bounds on the profit function in EP's A.3 imply a finite set of possible states, so we can compute and store equilibrium policies (at machine accuracy) for each one of them (Section 3).

(iii) states that if $s^t = (s_t, s_{t-1}, \ldots, s_1)$, then $P[s_{t+1} = s' \mid s^t] = P[s_{t+1} = s' \mid s_t] \equiv Q[s' \mid s_t]$, and $Q[\cdot \mid \cdot]$ is ergodic. I.e., there is a unique positive recurrent class, say $R \subset S$, and no matter s_0, s_t will, in finite time, wander into R, and *once in R there is no probability of communicating outside of R*; i.e., we can analyze subgames from R without knowing policies for $s \notin R$ (Section 4).

2. "BACKWARD SOLUTION" METHODS

We now describe an algorithm for computing equilibrium that is closely related to the one in Pakes and McGuire (1994; henceforth PM), but is modified to make it more easily comparable to the stochastic algorithm introduced in the next section.[5]

Substitute (2) into (1) and then rewrite that Bellman equation as

$$(3) \qquad V(i,s) = \max_{\chi \in \{0,1\}} \Bigg\{ [1-\chi]\phi + \chi \sup_{x \geq 0} \Big[\pi(i,s) - cx + \beta \sum_{\nu} w(\nu; i, s) p(\nu \mid x) \Big] \Bigg\},$$

where

$$(4) \qquad w(\nu; i, s) \equiv \sum_{(\hat{s}'_i, \zeta)} V(i + \nu - \zeta, \hat{s}'_i + e(i + \nu - \zeta)) q[\hat{s}'_i \mid i, s, \zeta] \mu(\zeta).$$

[5] For more detail on the computation of this class of models see PM and Pakes (2000). For good general overviews of backward solution algorithms for computing fixed points, see Bertsekas (1995) and Judd (1999, Ch. 12).

The $\{w(\nu; i, s)\}$ are the continuation value of the firm conditional on the current year's investment resulting in a particular ν and the current state being (i, s). They are constructed by summing over the probability weighted outcomes for the competitors' states (the $\hat{s}_i'$) and the common shock (ζ).

Let V be the support of ν, and note that $\{w(\nu; i, s) : (\nu, i, s) \in V \times \Omega \times S\} \subset \mathscr{R}_+^{\mathbf{w}}$ determines all policies. Pick a $w \in W = \{w : w \in \mathscr{R}_+^{\mathbf{w}}\}$ and use (3) to compute the decision rules and value function generated by w at each (i, s), say $x^w = T_x w$, $\chi^w = T_\chi w$, and $V^w = T_V w$. These decision rules determine transition probabilities, say $q^w = T_q w$ as

$$(5) \qquad q^w[\hat{s}_i' = \hat{s}_i^* \mid i, s, \nu] \equiv \Pr\{\hat{s}_i' = \hat{s}_i^* \mid i, s, \zeta, \text{the policies generated by } w\}$$

(for a formal description of q^w, see equations 6 to 8 in EP). Now substitute V^w and q^w into (4) and compute $w^+ = T_w w$. If $w = w^+$, then the equilibrium conditions are satisfied for the policies, value functions, and transition probabilities generated by w. Consequently the search for equilibrium policies can be recast as a search for a $w \in W$ that satisfies the fixed point $w = T_w w$. Any such w will be denoted by w^* (i.e. we abuse notation slightly by not distinguishing between a particular, and the set of, equilibrium values).

A backwards solution technique for finding w^* holds an estimate of w in memory, circles through the $s \in S$ in some predetermined order, and uses T_w to update each component of $w(\cdot; \cdot, s)$. The algorithm continues iterating until $Tw^j \approx w^j$. Its computational burden is essentially the product of:

(a) the number of points evaluated at *each* iteration ($\#S$);

(b) the time per point evaluated; and

(c) the number of iterations.

Both (a) and (b) grow *rapidly* in the number of state variables. To examine how (a) increases, temporarily allow L state variables per firm and recall that $\bar{n}$ is the maximum number of firms ever active. The rate of increase in (a) depends on whether we increase L or $\bar{n}$. If $i_l = 1, \ldots, k$ for $l = 1, \ldots, L$, then absent further restrictions, $K = \#\Omega = k^L$, an exponential in L (Section 5 considers possible restrictions). Each of the $\bar{n}$ firms can only be at K distinct states, so $\#S \leq K^{\bar{n}}$. However the policy functions are exchangeable in the state variables of a firm's competitors so we do not need to differentiate between two vectors of competitors that are permutations of one another. As shown in Pakes (1994) this implies that the combinatoric

$$\binom{K + \bar{n} - 1}{\bar{n}}$$

is an upper bound for $\#S$; but for $\bar{n}$ large enough this bound is tight. The bound increases geometrically (rather than exponentially) in $\bar{n}$. *All* numerical results we discuss (both PM's and ours) impose symmetry. Still (a) increases geometrically in $\bar{n}$ and exponentially in L.

(b) is primarily determined by the time to calculate the expected value of future outcomes (the w above). Say m firms are active from s, and that there is positive probability on each of κ points for each of the $m-1$ competitors of each firm (κ typically grows exponentially in L). Then to compute the required expectation we sum over κ^m possible future states, so the average computational burden per point is proportional to $\Sigma f(m) m \kappa^m$, where $f(m)$ is the fraction of points with m firms active.[6]

3. A STOCHASTIC ALGORITHM

The stochastic algorithm is *asynchronous*; it only updates policies at a single $s \in S$ at each iteration (j), and as a result must update both this *location*, s^j, and its estimate of continuation values, w^j. As in the literature on *Q-Learning*,[7] these updates are made to mimic what would happen if (i) actual agents chose policies assuming w^j provided the correct evaluation of possible future outcomes, (ii) nature resolved the randomness conditional on those policies and chose an s^{j+1}, and (iii) the agents treated s^{j+1} as a random draw from the states that could have occurred and averaged its estimated value with those of the previous draws from s^j to update $w^j(s^j)$.

Thus the stochastic algorithm replaces the deterministic update operator T_w: $W \to W$, with a Markov transition kernel, say $\mathscr{Q}(\cdot, \cdot \mid s, w)$, which takes $S \times W$ into a probability distribution on $S \times W$. At iteration j, $\mathscr{Q}(\cdot \mid s^j, w^j)$ is used to simulate s^{j+1}, and s^{j+1} is then used to form a Monte Carlo update of the continuation values (the w^j). This generates sample paths for s which eventually stay within a recurrent subset of S, say R, and hence reduces the number of points that must be updated ((a) above) from $\#S$ to $\#R$. The Monte Carlo updates of the w^j reduce the computational burden per point (i.e. (b) above) from $m\kappa^m$ to m, but introduces simulation error in w. To average out that error we need a large number of iterations. So the use of $\mathscr{Q}$ reduces (a) and (b), but increases (c). However since neither $\#R$, m, nor the number of iterations are necessarily related to the dimension of the state space, the stochastic algorithm may eliminate the curse of dimensionality altogether.

Let $(w^j, s^j) \in W \times S$ be the jth iteration values of (w, s). Policies for the jth iteration for each i with $s_i^j > 0$ are chosen by substituting $w^j(\cdot : i, s^j)$ for $w(\cdot : i, s)$ into (4) and choosing $(x(\cdot, s^j : w^j), \chi(\cdot, s^j : w^j))$ to maximize iteration j's

[6] The computational burden of obtaining the optimal policies given this sum need not grow in m at all (it does not in our example). The cardinality of this summand could be reduced by using symmetry restrictions, but this would require us to find the probabilities associated with each unique s_i' vector, a computationally burdensome task.

[7] See Barto, Bradtke, and Singh (1995), especially Sec. 7.3, and the literature cited there. Bertsekas and Tsikilis (1996, chapter 5), point out that Q-Learning can be obtained from the "$TD(\lambda) - k$" class of algorithms, algorithms that use a $TD(\lambda)$ method for updating the value function and k-iterations between policy updates, by setting $\lambda = 0$, $k = 1$; they refer to it as asynchronous optimistic policy iteration (see also their references to the reinforcement learning literature). We thank John Rust for pointing us to the relationship between our algorithm and this literature.

estimated value, i.e. as the solution to

$$(6)\qquad V(i, s^j : w^j) \equiv \max_{\chi \in \{0,1\}} \left\{ [1-\chi]\phi + \chi \sup_x \left[\pi(i, s^j) - cx + \beta \sum_{\nu} w^j(\nu; i, s^j) p(\nu \mid x) \right] \right\}.$$

Similarly if the potential entrant pays x_e dollars to enter and, if it enters, becomes an incumbent in the next period at i_e minus the common shock ζ, then the entry policy, say $\chi_e^j(s^j) \in \{0,1\}$, is

$$(7)\qquad \chi_e^j(s^j : w^j) = 1 \Leftrightarrow \beta w^j(0; i_e, s^j + e(i_e)) > x_e,$$

where $e(i_e)$ is a K-vector that has one for its i_e element and zero elsewhere.

These policies determine $\mathscr{Q}(\cdot, \cdot \mid s^j, w^j)$. To draw an s^{j+1} from this $\mathscr{Q}$:

(i) draw the common shock ζ^{j+1} from $\mu(\cdot)$,

(ii) for each active incumbent (each i for which $\chi(i, s^j : w^j) = 1$ and each of the s_i^j firms active at i), draw ν^{j+1} from $p(\nu \mid x(i, s^j : w^j))$ and compute $i + \nu^{j+1} - \zeta^{j+1}$, and

(iii) if there is entry (if $\chi_e(w^j) = 1$), compute $i_e - \zeta^{j+1}$.

Then if s_r^{j+1} counts the number of outcomes from (ii) and (iii) equal to r,

$$s^{j+1} = \left[s_r^{j+1}; r \in \Omega \right].$$

To update $w^j(\nu; i, s^j)$ we first form the jth iteration's evaluation of being in the location determined by (i, ν) when all the other competitors locations and ζ are determined by their random draws, or

$$(8)\qquad V\left(i + \nu - \zeta^{j+1}, \hat{s}_i^{j+1} + e(i + \nu - \zeta^{j+1}) : w^j\right)$$

as defined in (6). Note that (8) is the current iteration's perception of the value of a random draw from the integral defining $w^j(\nu; i, s)$. Hence we increment $w^j(\nu; i, s)$ by a fraction of the difference between (8) and $w^j(\nu; i, s)$. More formally let $w(s) = [w(\nu; i, s);\ \nu \in V,\ i \in \Omega]$. Then:

- if $s \neq s^j$, then $w^{j+1}(s) = w^j(s)$ (or $w(s)$ is not updated),
- if $s = s^j$, then

$$(9)\qquad w^{j+1}(\nu; i, s^j) - w^j(\nu; i, s^j) = \alpha(j, s^j)\left\{ V\left[i + \nu - \zeta^{j+1}, \hat{s}_i^{j+1} + e(i + \nu - \zeta^{j+1}) : w^j\right] - w^j(\nu; i, s^j) \right\},$$

where the $\alpha(j, s) \in (0,1)$ are chosen: (i) to be a function of information available at iteration j, and (ii) to have a sum that tends to infinity and a squared sum that remains bounded as the number of times the point is hit tends to infinity. $\alpha(j, s)$ equal to the inverse of the number of past iterations for which $s^j = s$, a choice that implies that the $w^j(\cdot; i, s)$ are the *simple average* of past draws on the conditional continuation values of firms at (i, s), will satisfy these conditions.[8]

Let $\mathscr{W}(\nu; i, s^j : w^j)$ be the expectation of (8) conditional on w^j, and let $\eta^{j+1}(s)$ be the difference between (8) and this expectation. Then

$$w^{j+1}(s) - w^j(s) = \alpha(j, s)[\mathscr{W}(s : w^j) - w^j(s) + \eta^{j+1}(s)],$$

with $E[\eta^{j+1}(s) \mid w^j] = 0$. The update will tend to increase or decrease $w^j(\nu; i, s)$ according as the expectation of this random draw is greater or less than $w^j(\nu; i, s)$. At the equilibrium (i.e. at w^*), $\mathscr{W}(s : w^*) = w^*(\forall s)$, so once we are at w^* we tend to stay their. When w^j equals w^* our algorithm generates an ergodic Markov process for $\{s^j\}$ and hence will, in a finite number of iterations, wander into the recurrent class (R) and stay in R thereafter.

We have shown how to update both w^j and s^j (the Appendix has more detail). We still need a rule for when to stop the iterations. Since we only obtain accurate policies on $R \subset S$, our stopping rule mimics a definition of equilibrium on subsets of S rich enough to enable us to analyze subgames from those subsets.

3.1. *Equilibrium Policies and Stopping Rules*

For any function f with domain S, and any $S^* \subset S$, let $f \mid S^*$ denote the restriction of f to S^*. Our algorithm eventually focuses in on an $S^{**} \subset S$. We say $w \mid S^*$ generates policies for $S^{**} \subset S^*$ if it contains the information needed to calculate $x(\cdot, s : w)$, $\chi(\cdot, s : w)$, and $\chi_e(s : w)$ for all $s \in S^{**}$. For $w \mid S^*$ to enable us to analyze subgames from S^{**} it must contain this information for each element of all possible sequences, $\{s_\tau\}_{\tau=t}^{\infty}$, for all $s_t \in S^{**}$.

DEFINITION: $w \mid S^*$ *generates equilibrium policies for subgames from* S^{**} iff $w \mid S^*$ generates policies for S^{**} and:

(d.i) $\inf_{\tau \geq t} \Pr\{s_\tau \in S^{**} \mid s_t, w\} = 1$, $\forall s_t \in S^{**}$, and,

(d.ii) the policies generated by $w \mid S^*$ satisfy the equilibrium conditions (6a to 6d in EP (1995)) $\forall s \in S^{**}$.

(d.i) insures that $w \mid S^*$ generates subgames from S^{**} while (d.ii) insures that those subgames are Markov Perfect equilibria.

The next theorem provides sufficient conditions for $w \mid S^*$ to generate equilibrium policies for subgames from S^{**} and is the basis for our stopping rule (condition (t.i) below insures (d.i) above, while (t.ii) insures (d.ii)). Write $\tilde{s} \rightarrow^w r$ iff the policies generated by w imply a positive probability of moving from $\tilde{s}$ to r (of $\tilde{s}$ communicating with r). Even if (d.i) and (d.ii) are satisfied, there will be $\tilde{s} \in S^{**}$ that could communicate with $r \notin S^{**}$ if feasible (though not optimal) policies are followed. The set of such r determine $\sim S^{**} \cap S^*$, and to check for

[8]The conditions are Robbins and Monroe's (1951) regularity conditions. Ruppert (1991) considers their importance and optimal weighting schemes (see also Section 3.2).

equilibrium policies on S^{**} we need estimates of $w(s)$ on $\sim S^{**} \cap S^{*}$ (the algorithm only produces accurate estimates of $w(s)$ for $s \in S^{**}$). Condition (t.ii)(b) states that all we require of the $w(s)$ for $s \in \sim S^{**} \cap S^{*}$ is that these $w(s) \geq w^{*}(s)$ (this insures that if a policy is not chosen it is indeed not optimal), so we obtain them elsewhere.

THEOREM 1: *Assume that for a $w \in W$ there is an $S^{**} \subset S^{*} \subset S$ such that $w \mid S^{*}$ generates policies $\forall s \in S^{**}$ and*:

(t.i) *if $\exists s \in S^{**} \rightarrow^{w} s'$, then $s' \in S^{**}$; and*

(t.ii) (*a*) *$\forall s \in S^{**}$ and $v \in V$, if $s_i > 0$,*

$$w(\nu; i, s) = \sum_{\tilde{s}'_i, \zeta} V[i + \nu - \zeta, \tilde{s}'_i + e(i + \nu - \zeta) : w] q^{w}(\tilde{s}'_i \mid i, s, \zeta) \mu(\zeta).$$

(*b*) *$\forall s \in \sim S^{**} \cap S^{*}$, $\exists w^{**}(s) \geq w^{*}(s)$, such that $w(s) \geq w^{**}(s)$.*
Then $w \mid S^{*}$ generates equilibrium policies for subgames from S^{*}.

PROOF: From (t.i), $q_{ij}^{w} = 0$ whenever $i \in S^{**}$ but $j \notin S^{**}$. An inductive argument shows that if Q has this property, then so does $Q^{\tau} \equiv Q Q^{\tau - 1}$, $\forall \tau \geq 1$. Thus $\forall \tau \geq t$, $s_t \in S^{**} \Rightarrow \Pr\{s_\tau \in S^{**} \mid s_t, w\} = 1 \Rightarrow \lim_{T \to \infty} \inf_{t \leq \tau < T} \Pr\{s_\tau \in S^{*} \mid s_t, w\} = 1$, which proves (d.i). To prove (d.ii), let $\tilde{w}(s) = w(s)$ if $s \in S^{**}$ and $\tilde{w}(s) = w^{*}(s)$ if $s \in \sim S^{**} \cap S^{*}$. Since w^{*} satisfies (t.ii), it is straightforward to check that the policies generated by $\tilde{w} \mid S^{*}$ satisfy the equilibrium conditions [6a to 6d in EP] $\forall s \in S^{**}$. So it suffices to show that $[\chi(\cdot, s : w), x(\cdot, s : w), \chi_e(s : w)] = [\chi(\cdot, s : \tilde{w}), x(\cdot, s : \tilde{w}), \chi_e(s : \tilde{w})]$, $\forall s \in S^{**}$.

Take $s \in S^{**}$. If $\chi(i, s : w) = 1$ and $\tilde{s}'_i$ has positive $q^{w}[\cdot \mid i, s, \zeta]$ probability, then $\tilde{s}'_i + e(i - \zeta) \in S^{**}$, so $w(0; i, s) = \tilde{w}(0; i, s)$ by (t.ii)(a). Similarly, since the support of ν is independent of x, if $x(i, s : w) > 0$ then $w(\nu; i, s) = \tilde{w}(\nu; i, s)$, and if $\chi_e(s : w) = 1$ then $w(0; i_e, s + e(i_e)) = \tilde{w}(0; i_e, s + e(i_e))$. The remaining cases are: (a) $\chi(i, s; w) = 0$ (or $\chi_e(s : w) = 0$); and (b) $\chi(i, s : w) = 1$ but $x(i, s : w) = 0$.

(a) $\Rightarrow \phi \geq \sup_x[\pi(i, s) - cx + \beta \Sigma_\nu w(\nu; i, s) p(\nu \mid x)] \geq \sup_x[\pi(i, s) - cx + \beta \Sigma_\nu \tilde{w}(\nu; i, s) p(\nu \mid x)] \Rightarrow \chi(i, s \mid \tilde{w}) = 0$, where the second inequality follows from (t.ii)(b) (to prove $\chi_e(s : w) = 0 \Rightarrow \chi_e(s; \tilde{w}) = 0$ substitute x_e for ϕ here).

(b) $\Rightarrow 0 \geq \sup_x\{\Sigma_\nu[w(\nu; i, s) - w(0; i, s)] p(\nu \mid x) - cx\} \geq \sup_x\{\Sigma_\nu[\tilde{w}(\nu; i, s) - w(0; i, s)] p(\nu \mid x) - cx\} \Rightarrow x(i, s \mid \tilde{w}) = 0$, where the first inequality follows from the optimality condition for x, the second from (t.ii)(b), and $\Rightarrow$ from $\chi(i, s : w) = 1 \Rightarrow w(0; i, s) = \tilde{w}(0; i, s)$. *Q.E.D.*

Our stopping rule is based on checking the two conditions of Theorem 1 for a w generated by the algorithm. To perform the check we need: a candidate for $S^{**}(w)$, and a $w^{**}(s) > w^{*}(s)$ for $s \in \sim S^{**} \cap S^{*}$. To obtain $S^{**}(w)$ note that any recurrent class of the process generated by $Q[\cdot, \cdot \mid w]$ satisfies the Theorem's condition (t.i). Further $\forall w \in W$ the process generated by $Q[\cdot, \cdot \mid w]$ is a finite state Markov chain, and all such chains have at least one recurrent class (see Freedman (1983, Ch. 1). Thus to obtain $S^{**}(w)$, use $Q[\cdot, \cdot \mid w]$ to simulate a sequence $\{s_j\}$, collect the set of states visited at least once between $j = J_1$ and

$j = J_2$ ($S^{J_2 - J_1}$), and set $S^{**}(w) = S^{J_1 - J_2}$. Provided both J_1 and $J_2 - J_1 \rightarrow \infty$, $S^{J_2 - J_1}$ will converge to a recurrent class of $Q[\cdot, \cdot \mid w]$, thus satisfying our condition (t.i), and our test need only check condition (t.ii)(a) on this set. That check requires a $w^{**}(s) \geq w^{*}(s)$ for $s \in \sim S^{**} \cap S^{*}$ but, as we note below, our initial condition (w^1) is greater than w^*, so setting $w^{**} = w^1$ will do.

3.2. *Details of the Algorithm*

A number of more detailed choices still need to be made (e.g. w^1 and the $\{\alpha(j, s)\}$). We begin with some results from a related artificial intelligence literature that both help make those choices and clarify the issues surrounding the convergence of our algorithm.

The stochastic algorithm generates a random sequence $\{w^j\}$, whose "increment," $V(\cdot \mid w^j) - w^j$, has a distribution, conditional on information available at iteration j, determined by w^j. We look for a zero root of the conditional expectation of this increment (this defines a w^*). Robbins and Monroe (1951) introduced a scalar version of a similar problem and provided conditions that insured $w^j \rightarrow w^*$ in mean square.[9] That paper stimulated a literature applying stochastic iterative solution techniques to an assortment of problems (see Ruppert (1991) for an accessible review). The branch most closely related to our problem is often referred to as reinforcement (or machine) learning (see Bertsekas and Tsikilis (1996) and the literature they cite).

The reinforcement learning literature typically deals with: (i) single agent dynamic programming problems in which, (ii) agents choose controls from a finite set of possible values, and (iii) if the algorithm is asynchronous, all points in the state space are recurrent (i.e. $S = R$). The use of discrete controls ((ii) above) leads to algorithms that simulate different future values for different choices of those controls. Instead we simulate values for alternative outcomes (our $w(\cdot; i, s)$), and choose a *continuous* control that determines the probability of those outcomes.[10] Still the algorithms are similar and convergence proofs related to those used in the machine learning literature can be adapted to the *single agent* version of our problem *when* $S = R$. These convergence proofs are of two types; the first is based on the existence of a smooth potential (or Lyapunov) function, f: $\mathscr{R}^{\mathrm{w}} \rightarrow \mathscr{R}$, which "looks inward" (has a gradient whose

[9]Blum (1954), generalizes to the finite dimensional case and almost sure convergence. Robbins and Monroe study the case where both the function $V(\cdot \mid w^j)$ and the family of conditional distributions may be unknown ("nature" generates the needed draws). In our case we can construct $q^w(\cdot \mid \cdot)$ and compute the expectation of $V(\cdot \mid w^j) - w^j$, but both these tasks are computationally burdensome (especially for large state spaces). We can, however, obtain random draws from $q^w(\cdot, \cdot)$ and evaluate $V(\cdot \mid w)$ easily.

[10]As noted, one advantage of our choices is that they imply a finite state space. Also the randomness in outcomes conditional on x implies that we need only experiment with policies that maximize the iteration's estimated values, and the availability of first order conditions helps find those policies.

dot product with the expected increment is negative), and the other is based on the operator T_w (defined in Section 2) being a contraction mapping.[11]

Now consider the $S = R$ assumption. The advantage of asynchronous algorithms when $S = R$ stems from the fact that in the asynchronous case the frequency with which a particular point is updated tends to the probability of that point in the ergodic distribution, and the precision of the estimates associated with the point increases in this frequency. Thus provided the estimates that will be used intensively are associated with points with more weight in the ergodic distribution, the asynchronous procedure will provide relatively precise estimates of intensively used points and will not waste much time on points that are rarely used. This advantage is likely to be even more important when R is small relative to S, and there is a literature on algorithms that replace the $R = S$ condition with other requirements. These include using a $w^1 \geq w^*$, since then all feasible policies will be sampled, and an updating procedure that maintains this inequality (this requires us to modify the way we compute continuation values and increases computational burden per point; see the Appendix to Barto, Bradtke, and Singh (1995)).

Moving from single agent problems to Markov Perfect Games is yet more difficult. The games we analyze do not produce a T_w that is a contraction mapping, and we have not been able to find a smooth potential function for our problem. Hence like other algorithms in use for computing fixed points that are not contractions (see Judd (1999)), we cannot insure convergence a priori. Still we can use a $w^1 > w^*$ and we can keep track of whether the sufficient conditions for convergence given by our theorem are satisfied.

Detailed Choices. To complete the description of the algorithm we need to choose initial conditions, the weights for the iterative procedure (the $\alpha(j,s)$ in equation (9)), the frequency with which we construct an S^{**} and test condition (t.ii)(a), and the norm for that test. We consider each of these in turn.

We tried two w^1's; (i) the value of $w(\cdot)$ from the one firm problem for the i of the (i,s) couple we are after, and (ii) $\pi(i,s)/(1-\beta)$. Since (i) does not mimic $V(i,s)$ when Σs_i is large, we expected the relative performance of $\pi(i,s)/(1-\beta)$ to improve with market size, but even at small market sizes it did quite a bit better.[12]

The stochastic approximation literature has a discussion of efficient choices for $\{\alpha(j,s)\}$. These are typically functions of j and the number of times the point s has been visited by iteration j, say $h^j(s)$. Since later iterations' valuations are likely to be closer to w^* than those from early iterations, efficient

[11] For more detail, see Bertsekas and Tsikilis (1996, Ch. 4); (Ch. 7 contains an extension to zero sum games). Typically a weighted maximum norm is used and T_w can have a unitary modulus of contraction provided it has a unique fixed point and generates bounded $\{w^j\}$ sequences.

[12] Both are easy to calculate (since nothing is in memory for a point the first time it is visited, $\pi(i,s)$ has to be calculated then anyway; see the Appendix). EP prove that $V(i) \geq V(i,s)\ \forall (i,s) \in \Omega \times S$, whereas though our examples had $\pi(i,s)/(1-\beta) \geq V(i,s)\ \forall (i,s) \in \Omega \times S$, this condition depends on the primitives used. Note that our discussion assumes that we do not have information on $\{w(\cdot)\}$ for a close (though not identical) set of primitives; else we might use them.

weighting schemes down-weight early outcomes (see Ruppert (1991)). The simple procedure we used started with our initial conditions and $\alpha(j,s) \equiv (1+h^j(s))^{-1}$, but after a million iterations was restarted with initial values given by the final values from the prior million and $h^j(s)$ reset to 1 for those s visited during the million iterations. We iterated on this 'averaging' procedure several times before going to one long run.

The long run was interrupted every million iterations to perform the test we now describe, and stopped when the test criteria were satisfied. When j was a test iteration, we set $S^{**}(j)$ equal to the set of points visited during the last million iterations and calculated firm values for all active firms and the potential entrant at each such $s \in S^{**}(j)$ twice: once as in equation (6) [labelled $V(i,s \mid w^j)$], and once explicitly summing over the probabilities of those values producing

$$\begin{aligned} &V^*(i,s \mid w^j) \\ &\quad \equiv \max_\chi \Big\{ [1-\chi]\phi + \chi \sup_x \Big[\pi(i,s) - cx \\ &\qquad\qquad + \beta \sum V(i', \hat{s}_i' \mid w^j) p(i' \mid i,x,\zeta) q^{w^j}(\hat{s}_i' \mid i,s,\zeta) \mu(\zeta) \Big] \Big\}. \end{aligned}$$

Our test consists of comparing V to V^*. As in the reinforcement learning literature we are more concerned with precision at frequently visited points so a weighted Euclidean norm of $V - V^*$, with weights proportional to the empirical distribution of visits in the long run, is used for the test.[13]

As noted we are not able to compute $V^*(i,s \mid w^j)$ for all $s \in S^{**}(j)$ solely from $w^j \mid S^{**}$ (we require $V(i,s \mid w^j)$, and hence $w^j(s)$, for some $s \notin S^{**}(j)$ that could communicate with $S^{**}(j)$ if feasible but nonoptimal policies were followed). We began by substituting $w^1(s)$ for $w^j(s)$ for those s. However it was clear from the earlier runs that the weight assigned to $V(i,s \mid w^j)$ for $s \notin S^{**}$ was so small that a number of procedures for choosing the needed $w(s)$ led to the same results, so for computational ease our later runs set the probability of an s whose $w(s)$ was needed and not in memory to zero and renormalized so the remaining probabilities summed to one.[14]

Possible Problems. Note that the computational burden of the *test* does go up *exponentially* in the number of state variables and that this is the *only* part of our algorithm that has this property. Also the use of our norm implies that we may stop with imprecise estimates at infrequently visited points. To mitigate any resultant problems the output of the algorithm includes a count of how many

[13]Critical values are given below. Note that we have modified the formal test in two ways: (i) for efficiency we use the iterations needed to construct S^{**} to improve the estimate of w, and (ii) we perform the tests on the value functions per se (instead of on w) as this makes diagnostics easier. Also, to ease storage requirements, before the test we discarded all those points that were not used since the last test.

[14]A similar point, i.e. that the imprecision in estimates associated with points of small probability tend to have little effect on the precision at other points, is perhaps the most important numerical finding of the reinforcement learning literature.

times each point has been visited. If the analyst needs policies from an infrequently visited s, one can either use local restart procedures to obtain more precise estimates in the required neighborhood, or revert to pointwise calculations that use the reliably computed values as terminal values for their locations.

4. NUMERICAL RESULTS

We begin with evidence on the precision of the stochastic algorithm and then consider its computational burden. All our calculations are for the differentiated product model analyzed in PM (1994), an example published before the stochastic algorithm was available.

Table I compares the value and policy functions computed by PM to those generated by a stochastic algorithm that averaged after each of the first ten million iterations, and then ran uninterrupted for an additional ten million. PM's model had $\nu \in \{0,1\}$ and $p(\nu = 1 \mid x) = ax/(1+ax)$;[15] so we compare the results on $V(\cdot)$ and $p(\nu = 1 \mid x(\cdot))$. The entries appearing in the table weight the estimates of $V(\cdot)$ and $p(1 \mid x(\cdot))$ at each (i,s) by the number of times the different combinations were visited in the long run.

This table shows very little difference in the results from the two algorithms. Interestingly this is in spite of the fact that due to computational costs PM set $\bar{n} = 6$ and the stochastic algorithm finds .2% of the points in the ergodic distribution had seven active firms.[16]

To answer the question of whether the two sets of results had different implications for other statistics of interest, we substituted the stochastic algorithm's policies into the simulations PM used to characterize their dynamic equilibrium, and recalculated PM's descriptive tables. Our simulation used

TABLE I

EXACT VS. STOCHASTIC FIXED POINT[a]

	Mean	Standard Deviation	Correlation
$p(\nu = 1 \mid x_{(i,s)})$ (stochastic)	.6403	14.46	.9993
$p(\nu = 1 \mid x_{(i,s)})$ (exact)	.6418	14.43	⋯
$V(i,s)$ (stochastic)	12.40	643.8	.9998
$V(i,s)$ (exact)	12.39	639.6	⋯

[a]The entries are weighted sample statistics with weights equal to visit frequencies.

[15]A computational advantage of this choice is that it allows for an analytic solution for the optimal x^j as a function of w^j. Note that in applied work we generally can, by choice of number of decision periods per data period, translate the two point distribution of outcomes per decision period into a very rich family for increments in ν over data periods (though β must be chosen relative to the decision period).

[16]The only place the stochastic algorithm uses a bound on $\bar{n}$ is in the procedure that stores and retrieves information (see the Appendix). Since this can be increased at little cost (without computing all possible values and policies), we set $\bar{n} = 10$ for all runs.

TABLE II

"Discrete" Statistics from Simulation Runs

	(1)	(2)	(3)	(4)	(5)
		Stochastic Fixed Point			
	Exact Fixed Point	Average 100 runs	Standard Deviation 100 runs	Max/Min 100 runs	Standard Deviation (policies constant)
Percentage of periods with n firms active					
$n =$					
3	61.9	58.3	02.9	64.0/49.7	02.8
4	34.4	33.7	02.6	40.0/27.7	02.5
5	03.2	06.3	00.8	08.1/04.7	00.7
6	00.5	01.5	00.3	02.4/00.9	00.3
7	00.0	00.2	00.1	00.3/00.0	00.1
8	00.0	00.0	00.0	00.0/00.0	00.0

different starting values (s_0) and different random draws than PM's. Still our results on the *distributions* of the *continuous* valued random variables (i.e., the one firm concentration ratios, the price cost margins, the realized firm values, . . .) were virtually *identical* to those published in PM. On the other hand, as columns (1) and (2) of Table II illustrate, there were noticeable differences in the distribution of the number of firms active in different periods.

The number of firms active is a discontinuous function of *both* the estimates of the value function, and of the values of the random draws (including s_0). To separate out the differences between columns (1) and (2) caused by differences in estimated policies we ran the stochastic algorithm one hundred times, and simulated 10,000 periods of output from each estimate of w. We then compared the variance in the distribution of the number of firms active (column (3)) to the variance we obtained when we ran 100 independent simulations from the *same* w (column (5)). The differences between the squares of columns (3) and (5) estimate the fraction of the variance in the distribution of the number of firms active caused by differences in policies. It is too small to be of much concern.[17] There is *no* evidence of meaningful differences in the policy outputs from the "exact" and the stochastic algorithms.

Moving to an analysis of computational burden, note that $\bar{n}$ for PM's problem is largely determined by their M (the number of consumers serviced by the market). Table III pushes M up from PM's initial $M = 5$ by units of 1 until $M = 10$. Each run of the algorithm averaged after each of the first seven million iterations, and then began a long run which was interrupted every million iterations to run the test. The algorithm stopped only if the weighted correlation between our test value functions was over .995 and the difference between their weighted means were less than 1%. The bottom panel of the table provides

[17]Another way to see this is to note that, except possibly at the upper tail of the distribution of entrants, the standard errors in column (3) are large relative to the differences between columns (1) and (2); and since PM incorrectly set $\bar{n} = 6$, we do not expect the upper tails to be comparable.

TABLE III

COMPARISONS FOR INCREASING MARKET SIZE[a]

$M =$	5	6	7	8	9	10
Percentage of equilibria with n firms active						
$n =$						
3	58.3	00.8	00.0	00.0	00.0	00.0
4	33.7	77.5	48.9	04.4	00.7	00.1
5	06.3	16.8	41.4	62.3	33.0	07.2
6	01.5	04.2	07.3	25.0	44.3	41.8
7	00.2	00.6	02.2	06.5	15.3	34.3
8	00.0	00.1	00.2	01.7	05.9	13.1
9	00.0	00.0	00.0	00.0	00.8	03.5
10	00.0	00.0	00.0	00.0	00.0	00.0
Average n						
	3.43	4.26	4.64	5.39	5.95	6.64
Minutes per Million Iterations						
	5.5	6.5	7.5	8.6	10	11
Minutes per Test						
	3.6	8.15	17.1	42.8	100	120
Number of Iterations (millions)						
	$7 + 5$	$7 + 2$	$7 + 21$	$7 + 4$	$7 + 9$	$7 + 3$
Number of Points (thousands)						
	21.3	30.5	44.2	68.1	98.0	117.5

[a]All runs were on a Sun SPARCStation 2.

statistics that enable us to assess how the computational burden changed with market size. The top panel provides the distribution of the number of firm's active from a 100,000 period simulation and the estimated w.

The number of points in the last row refers to the number of points visited at least once in the last million iterations. This will be our approximation to the size of the recurrent class ($\#R$). There were 21,300 such points when $M = 5$. As expected $\bar{n}$ increases in M. However $\#R$ *does not* increase geometrically in $\bar{n}$. The top part of the panel makes it clear why; when we increase M the number of distinct points at which there are a large number of firms active does increase, but the larger market no longer supports configurations with a small number of firms. Indeed though the function relating $\#R$ to M seems initially convex, it then turns concave giving the impression that it may asymptote to a finite upper bound.

Comparisons to the backward solution algorithm will be done in terms of both memory and speed. The ratio of memory requirements in the two algorithms is effectively $\#R/\#S$. $\#R/\#S \approx 3.3\%$ when $\bar{n} = 6$, about .4% when $\bar{n} = 10$ (then $\#S \approx 3.2 \times 10^7$), and would decline further for larger $\bar{n}$.

The CPU time of the stochastic algorithm is determined by the time per iteration, the number of iterations, and the test time. The time per iteration is essentially a function of the number of firms active so that the ratio of the average number of active firms to the time per million iterations only varied between 1.53 and 1.65 minutes. The number of iterations until our test criteria was satisfied varied quite a bit between runs, but did not tend to increase in M.

Thus *absent* test times, the average CPU time needed for our algorithm seems to grow *linearly* in the average number of firms active. Given the theoretical discussion this is as encouraging a result as we could have hoped for.

As noted the *test* times do grow exponentially in the number of firms and in our runs they rose from about 3 minutes when $M = 5$ to over two hours when $M = 10$. By $M = 10$ the algorithm spends ten times as much time computing the test statistic after each million iterations as it spends on the iterations themselves. More research on stopping procedures that make less intensive use of our test seems warranted.[18]

Comparing our CPU times to those from backward solution algorithms, we find that when $\bar{n} = 6$ the stochastic algorithm took about half the c.p.u. time (about a third if one ignores test time). When $\bar{n} = 10$ even the most optimistic projection for the backward techniques leads to a ratio of c.p.u. times of .4% (.09% without test times). If $\bar{n}$ grew much beyond that, or if there were more than one state variable per firm, the backward solution algorithm simply could not be used (even on the most powerful of modern computing equipment). In contrast, we have analyzed several such problems on our workstation.

5. RELATED RESULTS

As noted, our algorithm's computational advantages should be particularly large for models in which there are a large number of state variables and $\#R \ll \#S$. Our example focussed on the relationship between $\bar{n}$, $\#R$ and $\#S$. Applied I.O. models often also require a large number of states per firm (or L).

We noted that without further restrictions $\#S$ grows exponentially in L.[19] However, our experience is that in I.O. models $\#R$ tends to grow much slower in L then does $\#S$. For example in differentiated product models where the state vector details different characteristics of the products (e.g., the size, mpg, and hp of cars), the primitives often indicate that certain combinations of characteristics are never produced in equilibrium (e.g. large cars with a high mpg, or small cars with a low mpg). Alternatively in locational models where there is an initial locational choice and then a plant specific cost of production (or quality of product) that responds to investments, $\#R$ tends to be linear in the number of locations.[20]

[18] Even procedures that screen on necessary conditions for equilibria before going to the full test can be quite helpful. We have used the fact that if $\{w\}$ is close to $\{w^*\}$, then the changes in $\{w\}$ ought to be a martingale to compute such screens.

[19] Often further restrictions are available. Models with multi-product firms, i.e. in which there may be only one state variable per product but there are many products per firm, are an important example (see Gowrisankaran (1999)).

[20] Still some problems will be too large for our algorithm. Recent papers in the AI literature that combine reinforcement learning techniques (similar to those used here) with approximation methods in the spirit of those described in Judd (1999) should then prove helpful (see Ch. 6 in Bertsekas and Tsikilis (1996), and the related work on approximations for single agent structural estimation problems by Keane and Wolpin (1994) and Rust (1997)).

The usefulness of our techniques for computing Nash equilibria will also vary with aspects of the problem not discussed here. For example to compute the equilibria of models in which behavior depends on values "off the equilibrium" path, as is true in many models with collusion, the algorithm will have to be modified so that it samples from the relevant nonequilibrium paths. On the other hand the fact that the algorithm mimics what would happen if agents were actually choosing policies based on our (fairly intuitive) learning rules, nature selected a market outcome from the distribution determined by their actions, and the agents used that outcome to update their estimate of continuation values, makes it possible that related techniques will be useful where standard Nash equilibrium behavior is not.[21]

Finally we note that we have *never* had a run of the stochastic algorithm that did not converge. This is in *marked contrast* to results using backward solution algorithms where convergence problems do occur and one has to resort to an assortment of costly procedures to overcome them (see PM).

Department of Economics, 117 Littauer Center, Harvard University, Cambridge MA 02138-3001, U.S.A.; ariel@ariel.fas.harvard.edu

and

Economic Growth Center, Yale University, New Haven CT 06520, U.S.A.; paul@oberon.econ.yale.edu.

Manuscript received December, 1996; final revision received August, 2000.

APPENDIX: OUTLINE OF A COMPUTER PROGRAM

Let $j \in Z^+$ index the iteration of the algorithm, and s^j provide its location at iteration j; $s^j \equiv (s_1^j, s_2^j, \ldots, s_k^j)$, so s_i^j is the number of active firms with state vector equal to i at iteration j. Let $h_s(j)$: $Z^+ \to Z^+$ denote the number of times location s has been hit prior to iteration j (note that $h_s(j) = 0$ is possible). Finally $V(s) \equiv \{V(i,s); \forall i \text{ with } s_i > 0\}$ will be called the permutation cycle of value functions at s (similarly for profits, policies, ...).

The algorithm calls three subroutines, and we describe them first. Next we note what is in memory at iteration j, and we conclude by showing how iteration $j+1$ updates that information.

Subroutines Needed

The first calculates the permutation cycle of profits for a given s, or $\pi(s)$ (see Section 1). This subroutine is called when a point is hit for the first time, and the calculated values are stored in memory. The second subroutine calculates initial values for the w's associated with an s that is visited for the first time, $\{w^1(\nu; s),\ \nu \in V\}$. As noted one possibility is to use $\{\pi(i+\nu, s)/(1-\beta),\ \nu \in V, \forall i \text{ with } s_i > 0\}$, in which case the initial conditions are taken directly from the profit calculation.

[21] Note that though the interpretation of our algorithm as a learning algorithm is a useful pedagogic device, its empirical validity depends on several issues. These include the method by which information on past outcomes is made available to the agents currently active, and the number of updates possible per unit of time.

The third subroutine stores and retrieves the information associated with alternative values of s. We used a trinary tree for this purpose. This starts by searching for the location of the active firm with the highest "i". The search from any given i can move to: (i) a larger i, (ii) a smaller i, or, having found the correct i, (iii) begin a search for the i of the active firm with the next highest i. Note that the depth of this search grows logarithmically in the number of points stored. There are alternative variants of trees, and alternative storage and retrieval schemes (such as hashing functions) that might prove more efficient.

In Memory at Iteration j

For simplicity assume $\nu \in \{0,1\}$. Then for each s with $h_s(j) \geq 1$, we have in memory the vector

$$\{w^j(0,s), w^j(1,s), \pi(s), \text{ and } h_s(j)\}.$$

There is *nothing* in memory for points that have not been visited.

Step 1: *Calculating Policies at* s^j.

Assume we are at $s^j = (s_1^j, \dots, s_k^j)$ and let $p(x)$ be the probability that $\nu = 1$. There are two cases to consider, $h_s(j) \geq 1$ and $h_s(j) = 0$.

If $h_s(j) \geq 1$, then for all $i \in \Omega$ with $s_i^j > 0$, calculate the couple $(\chi, x) \in \{[0,1], R^+\}$ that solves

$$\max(\chi)\{(1-\chi)\phi$$
$$+\chi \sup_x[\pi(i,s^j) - x + \beta p(x) w^j(1; i, s^j) + \beta(1-p(x)) w^j(0; i, s^j)]\}.$$

These policies, say $x^j(i,s^j) = x(i, s^j \mid w^j)$ and $\chi^j(i,s^j) = \chi(i, s^j \mid w^j)$, maximize future value if the jth iteration's conditional valuations (the w^j) are correct.

If $h_s(j) = 0$, call the subroutines that compute profits and initial conditions. Then calculate the policies that maximize the analogous expression with $\{w^1(1;s),\ w^1(0;s)\}$ substituted for $\{w^j(1;s), w^j(0;s)\}$.

Step 2: *Updating* $s(j)$.

Recall that a firm's state variable can take on only K values. Begin by setting a set of K counters to zero (these will count the number of firms at each different value of the state variable at iteration $j+1$).

Draw ζ^{j+1}, the change in value of the outside alternative, from $\mu(\zeta)$. Then, starting from the lowest value of i with $s_i^j > 0$, do the following. If $\chi^j(i,s^j) = 0$, skip all firms at s_i^j and go to s_{i+1}^j. If $\chi^j(i,s^j) = 1$, then for each of the s_i^j firms at i draw ν_i^{j+1} from $p[\cdot \mid x^j(i,s^j)]$, and up the counter at location $i + \nu_i^{j+1} - \zeta^{j+1}$ by one.

This determines exit and the next iteration's i for all incumbents that remain active. We need also to account for possible entry. Recall that in the simplest entry model a potential entrant pays x_e dollars to enter, and, if it enters, becomes an incumbent at location $i = i_e - \zeta^{j+1}$ in the next period. Thus the value of entry exceeds the cost of entry only if $\beta w^j(0; i_e, s^j + e(i_e)) > x_e$, where $e(i_e)$ is a K-vector that has one for its i_e element and zero elsewhere. If this condition is satisfied up the counter at $i_e - \zeta^{j+1}$ by one.

The vector of values from the set of counters (ordered from the lowest location) is s^{j+1}.

Step 3: *Updating* w^j.

If $h_s(j) = h_s(j+1)$, that is if location s was not hit at iteration j, then the data in memory for location s are not changed. If $h_s(j+1) = h_s(j) + 1$ we do update. Assume that $h_s(j) \geq 1$ and recall that

$$s^{j+1} - e[i + \nu_i^{j+1} - \zeta^{j+1}]$$

is the vector providing the outcomes of the random draws from location s for all but the ith firm. Then for $\nu = \{0, 1\}$,

$$w^{j+1}(\nu; i, s) = [h_s(j)/(h_s(j)+1)]w^j(\nu; i, s) + [1/(h_s(j)+1)]$$
$$\times V^j\{(i + \nu - \zeta^{j+1}, s^{j+1} - e[i + \nu_i^{j+1} - \zeta^{j+1}] + e[i + \nu - \zeta^{j+1}]\}.$$

The couple, $[i + \nu - \zeta_{j+1}, s^{j+1} - e[i + \nu_i^{j+1} - \zeta^{j+1}] + e[i + \nu - \zeta^{j+1}]$, provides the state that would have been achieved had the firm's own research outcome been ν, but those of its competitors and of the outside alternative been determined by the realizations of the simulated random variables from Step 2. Thus the $(j+1)$st estimate of the value of the research outcome ν is a weighted average of: (i) the iteration j estimate of that value [with weight equal to $h_s(j)/(h_s(j)+1)$], and (ii) the jth iteration's evaluation of the state determined by $i + \nu - \zeta(j+1)$, and the *actual realization* of the simulated research outcomes of all competitors. If $h_s(j) = 0$ use $\{w^1(0; i, s), w^1(1; i, s)\}$ for $\{w^j(0; i, s), w^j(1; i, s)\}$ and do the same calculation.

That completes the iteration of our algorithm (it now returns to Step 1 and continues). We note that most of the procedures used in the dynamic programming literature for improving the accuracy of updates could also be used in our Step 3 (see Ch. 6 of Bertsekas and Tsikilis (1996) and Judd (1999)).

REFERENCES

BARTO, A. G., S. I. BRADTKE, AND S. SINGH (1995): "Learning to Act Using Real-Time Dynamic Programming," *Artificial Intelligence*, 72, 81–138.

BERTSEKAS, D. (1995): *Dynamic Programming and Optimal Control*, Volumes 1 & 2. Belmont, MA: Athena Scientific Publications.

BERTSEKAS, D., AND J. TSITSIKLIS (1996): *Neuro-Dynamic Programming*. Belmont, MA: Athena Scientific Publications.

BLUM, J. (1954): "Multivariate Stochastic Approximation Methods," *Annals of Mathematics and Statistics*, 25, 737–744.

ERICSON, R., AND A. PAKES (1995): "Markov Perfect Industry Dynamics: A Framework for Empirical Work," *Review of Economic Studies*, 62, 53–82.

FREEDMAN, D. (1983): *Markov Chains*. New York: Springer Verlag.

FUDENBERG, D., AND D. LEVINE (1999): *Learning in Games*. Cambridge: MIT Press.

GOWRISANKARAN, G. (1999): "Efficient Representation of State Spaces for Some Dynamic Models," *Journal of Economic Dynamics and Control*, 23, 1077–1098.

JUDD, K. (1999): *Numerical Methods in Economics*. Cambridge, Mass: M.I.T. Press.

KEANE, M., AND K. WOLPIN (1994): "The Solution and Estimation of Discrete Choice Dynamic Programming Models by Simulation and Interpolation: Monte Carlo Evidence," *Review of Economics and Statistics*, 76, 648–672.

LETTAU, M., AND H. UHLIG (1999): "Rules of Thumb versus Dynamic Programming," *American Economic Review*, 148–174.

MASKIN, E., AND J. TIROLE (1988): "A Theory of Dynamic Oligopoly: I and II," *Econometrica*, 56, 549–599.

——— (1994): "Markov Perfect Equilibrium," Harvard Institute of Economic Research Discussion Paper.

PAKES, A. (1994): "Dynamic Structural Models, Problems and Prospects," in *Advances in Econometrics*, *Proceedings of the Sixth World Congress of the Econometric Society*, ed. by J. J. Laffont and C. Sims. NY: Cambridge University Press.

——— (2000): "A Framework for Applied Dynamic Analysis in I.O.," NBER Discussion Paper #8024.

PAKES, A., AND P. MCGUIRE (1994): "Computing Markov-Perfect Nash Equilibria: Numerical Implications of a Dynamic Differentiated Product Model," *RAND Journal of Economics*, 25, 555–589.

ROBBINS, H., AND S. MONROE (1951): "A Stochastic Approximation Method," *Annals of Mathematics and Statistics*, 22, 400–407.

RUPPERT, D. (1991): "Stochastic Approximation," in the *Handbook of Sequential Analysis*, ed. by B. Ghosh and P. K. Sen. New York: Marcel Dekker Inc.

RUST, J. (1997): "Using Randomization to Break the Curse of Dimensionality," *Econometrica*, 65, 487–516.

SARGENT, T. (1993): *Bounded Rationality in Macroeconomics*. Oxford: Oxford University Press.